Nom: Prénom: Groupe:

Université de Bordeaux, Département Licence
4TPM205CC

Devoir surveillé du 07/03/2017
9h00-10h30

Algorithmique des tableaux

Le sujet comporte 5 pages. Aucun document n'est autorisé. Répondez directement sur le sujet.

Exercice 1 Pour chaque valeur de la variable `x` complétez le tableau suivant :

— en calculant la valeur de l'expression correspondante et

— en précisant les étapes de calcul nécessaires pour obtenir le résultat.

	Expression	Valeur	Justification (`x` pouvant valoir soit `True` soit `False`)
`x=8`	`x%2==0 or x>10`	`True`	`True or x` vaut `True` (evaluation paresseuse)
`x=7`	`x%2==0 or x<10`	`True`	`False or True` vaut `True`
`x=20`	`x>0 and x%3==0`	`False`	`True and False` vaut `False`
`x=15`	`x%2==0 and x%3==0`	`False`	`False and x` vaut `False` (evaluation paresseuse)
`x=4`	`not(x<0) or not(x%2==0)`	`True`	`True or x` vaut `True` (evaluation paresseuse)
`x=10`	`not(x%2==0 or x%3==0)`	`False`	`True or x` vaut `True` (evaluation paresseuse) not(`True`) vaut `False`
`x=4`	`not(x%2==0 and not(x%3==0))`	`False`	`True and True` vaut `True` not(`True`) vaut `False`

Exercice 2 On suppose que l'on dispose d'une fonction `pile()` qui simule un lancer d'une pièce. Si le résultat du lancer est *pile*, la fonction renvoie `True`. Si c'est *face*, elle renvoie `False`.

1. Écrire une fonction `lancer(k)` qui renvoie le nombre de lancers nécessaires (le nombre d'appels à la fonction `pile()`) pour que le nombre de *piles* moins le nombre de *faces* soit au moins `k`. On supposera `k` > 0 entier.
   Par exemple, si la suite des lancers de *piles* (P) ou *faces* (F) est

   F P F P P F P P ...

   alors `lancer(2)` devra renvoyer 8. Notez que sur la même suite de *piles* ou *faces*, `lancer(1)` aurait renvoyée 5.

   Solution 1 :

```
def lancer(k):
    p=0 # nombre de piles
    f=0 # nombre de faces
    n=0 # nombre de lancers
    while not (p-f>=k):   # aussi:  while p-f < k :
        if Pile():
            p += 1
        else:
            f += 1
        n += 1
    return n   # on peut se passer de n et renvoyer p+f
```

   Solution 2 : en enlevant le "not" et en remarquant que f=n-p

```
def lancer(k):
    p=0 # nombre de piles
    n=0 # nombre de lancers
    while 2*p-n<k:
        if Pile():
            p += 1
        n += 1
    return n
```

2. On souhaite modifier la fonction `lancer(k)` de façon à obtenir le nombre de lancers nécessaires pour que le nombre de *piles* moins le nombre de *faces* soit au moins `k` et que le nombre de *piles* soit un multiple de `k`. On supposera toujours `k` > 0 entier. Donner les modifications à apporter à la fonction de la question précédente.

   Modification de la solution 1 :

```
while not (p-f>=k and p%k==0):
```

   Modification de la solution 2 :

```
while 2*p-n<k or p%k>0:
```

Exercice 3 On définit la factorielle impaire d'un entier $n > 0$ comme le produit de tous les entiers positifs **impairs** inférieurs ou égaux à n. De plus, par convention, la factorielle impaire de 0 vaut 1.

1. Compléter (sans effectuer le calcul) :

La valeur de la factorielle impaire de 7 est : $1 \times 3 \times 5 \times 7$

Nom: Prénom: Groupe:

La valeur de la factorielle impaire de 10 est : $1 \times 3 \times 5 \times 7 \times 9$

2. Ecrire une fonction `factorielleImpaire(n)` qui calcule et renvoie la valeur de la factorielle impaire de n. On suppose $n \geq 0$.

Solution :

```
def factorielleImpaire(n):  # nb de multiplication : (n+1)//2
    f = 1
    for i in range (1, n+1, 2):
        f = f*i
    return f
```

3. On souhaite écrire une fonction `sommeFactoriellesImpaires(n)` qui calcule et renvoie la somme :

`factorielleImpaire(0)+factorielleImpaire(1)+···+factorielleImpaire(n-1)+factorielleImpaire(n)`

Écrivez une première version de `sommeFactoriellesImpaires(n)` en utilisant la fonction `factorielleImpaire` écrite précédemment.

Solution :

```
def sommeFactorielleImpaire(n):
    s = 0
    for i in range(n+1):
        f = factorielleImpaire(i)
        s = s + f
    return s
```

4. Combien de multiplications sont-elles nécessaires pour calculer `sommeFactoriellesImpaires(n)` ? Justifier.

Solution :

Nombre de multiplications : $O(n^2)$ obtenu en faisant la somme, pour `i` allant de 1 à n de `((i+1)//2)`, c'est à dire `(n//2)*(n//2+1)+(n%2)*(n+1)//2`

5. Écrivez une deuxième version `sommeFactoriellesImpairesBis(n)` en modifiant la version précédente pour calculer le résultat **sans** utiliser d'appel à la fonction `factorielleImpaire`.

Solution :

```
def sommeFactorielleImpaire2(n):
    s = 0
    f = 1
    for i in range(n+1):
        if i%2 ==1 :
            f = f*i
        s = s+f
    return s
```

6. Combien de multiplications sont-elles nécessaires pour calculer `sommeFactoriellesImpairesBis(n)` ? Justifier.

Solution :

Nombre de multiplications : $O(n)$ car le nombre exact est `n//2+n%2`

Exercice 4 Soit un tableau t de n entiers. On dit qu'un élément $t[i]$, $0 < i < n-1$ est un `pic` si et seulement si $t[i-1] < t[i] > t[i+1]$.
Ainsi par exemple, pour le tableau $t = [7, 8, 5, 6, 4, 10, 9]$, l'élément $t[1] = 8$ est un `pic` car $7 < 8$ et $8 > 5$.

1. Écrire une fonction `estPic(t, n, i)` qui vérifie si l'élément du tableau à l'indice `i` est un `pic`. Le tableau `t`, son nombre `n` d'éléments et un indice `i` sont passés en paramètre.
   Pour l'exemple ci-dessus, l'appel à la fonction `estPic(t, 7, 3)` renvoie `True`, et l'appel à la fonction `estPic(t, 7, 4)` renvoie `False`. Si `i` vaut 0 ou $n-1$, la fonction doit renvoyer `False`.
   Solution :

```
def estPic(t, n, i):
    if i==0 or i==n-1:
        return False
    return t[i] > t[i-1] and t[i] > t[i+1]
```

2. Écrire une fonction `nombreDePics(t, n)` qui renvoie le nombre de pics dans le tableau (contenant n éléments) passé en paramètre.
   Pour l'exemple ci-dessus, l'appel à la `nombreDePics(t, 7)` va renvoyer `3`.
   Solution :

```
def nombreDePics(t, n):
    nb = 0
    for i in range(1,n-1):
        if estPic(t, n, i):
            nb = nb + 1
    return nb
```

Exercice 5 Étant donnés deux tableaux s et t, on dit que s *est un sous-tableau de* t quand la séquence de valeurs dans s est une sous-séquence de celle dans t. Plus précisément, si $s = [s_0, s_1, \ldots, s_m]$ et $t = [t_0, t_1, \ldots, t_n]$, s *est un sous-tableau de* t quand $m \leq n$ et il existe une suite d'indices $i_0 < i_1 < \cdots < i_m \leq n$ tels que $s_0 = t_{i_0}, s_1 = t_{i_1}, \ldots, s_m = t_{i_m}$.

Par exemple, $[5, 3, 6]$ est un sous-tableau de $[4, 1, \underline{5}, 5, 6, \underline{3}, 2, 9, \underline{6}, 6, 4]$ et $[2, 5, 8, 9]$ est un sous-tableau de $[1, 5, \underline{2}, \underline{5}, 9, \underline{8}, 1, 1, \underline{9}, 2]$.

1. Écrire une fonction `sousTableau(s,m,t,n)` qui prend en entrée deux tableaux s et t contenant respectivement m et n éléments. La fonction devra retourner `True` quand s est un sous-tableau de t et `False` sinon.
   Solution :

```
def sousTableaux(s, m, t, n):
    if m>n:
        return False
    j = 0  #indice dans s
    i = 0  #indice dans t
    while j<m and i<n:
        if t[i]==s[j]:
            j+=1;
        i+=1
    return j==m
```

2. On utilise comme mesure de complexité le nombre de tests d'égalité effectués entre une valeur venant de s et une valeur venant t. Évaluez ce nombre (en fonction de m et n) pour votre fonction.
   Solution :
   Les tests d'égalité ont lieu dans la boucle `while` qui est exécutée au minimum m fois (cas où les m éléments de s coincident, dans l'ordre, avec les m premiers éléments de t) et au

plus n fois (cas où s est sous-tableau de t et $s_m = t_n$) . Un et un seul test a lieu à chaque tour de boucle. Puisque $m \leq n$, il y aura au maximum $O(n)$ tests d'égalité.